



UNIVERSITI PUTRA MALAYSIA

**EXTRACTING OBJECT ORIENTED SOFTWARE ARCHITECTURE
FROM C++ SOURCE CODE**

ALI HUSSEIN A. MRESA

FSKTM 2000 6

**EXTRACTING OBJECT ORIENTED SOFTWARE ARCHITECTURE
FROM C++ SOURCE CODE**

By

ALI HUSSEIN A. MRESA

**Thesis Submitted in Fulfilment of the Requirements for the
Degree of Master of Science in the Faculty of
Computer Science and Information Technology
Universiti Putra Malaysia**

October 2000



To the Soul of My Sisters Al-forjania and Soad

To My Parents

Abstract of thesis presented to Senate of University Putra Malaysia in fulfilment of the requirements for the degree of Master Science.

**EXTRACTING OBJECT-ORIENTED SOFTWARE ARCHITECTURE
FROM C++ SOURCE CODE**

By

ALI HUSSEIN A. MRESA

October 2000

Chairman: Abdul Azim Abdul Ghani, Ph.D.

Faculty: Computer Science and Information Technology

Software architecture strongly influences the ability to satisfy quality attributes such as modifiability, performance, and security. It is important to be able to analyse and extract information about that architecture. However, architectural documentation frequently does not exist, and when it does, it is often out of sync with the implemented system. In addition, it is not all that software development begins with a clean slate; systems are almost always constrained by the existing legacy code. As a consequence, there is a need to extract information from existing system implementations and reason architecturally about this information.

This research presents a reverse engineering tool VOO++ that will read an Object-Oriented C++ source code using UML notation in order to visualise its Class structure and the various relationships that may exist including, inheritance, aggregation, and dependency relationships based on the modified Cohen-Sutherland clipping algorithm.

The idea of clipping is reversed, instead of clipping inside the rectangle, the clipping is done outside the rectangle in terms of four directions (left, right, top, and bottom) and two points represent the centre point for each rectangle.

An Object-Oriented approach is used to design and implement the tool. Reverse engineering, design pattern, and graphics are the underlying techniques supplied. VOO++ aids an analyst in extracting, manipulating and interpreting the Object-Oriented static model information. By assisting in the reconstruction of static architectures from extracted information, VOO++ helps an analyst to redocument and understand architectures and discover the relationship between “as-implemented” and “as-designed” architectures.

Abstrak tesis yang dikemukakan kepada Senat Universiti Putra Malaysia bagi memenuhi keperluan untuk ijazah Master Sains.

**PENGHASILAN SENI BINA PERISIAN BERASASKAN
OBJEK DARI KOD SUMBER C++**

Oleh

ALI HUSSEIN A. MRESA

Oktober 2000

Pengerusi: Dr. Abdul Azim Abd Ghani, Ph.D.

Fakulti: Sains dan Pengajian Alam Sekitar

Seni bina perisian sangat mempengaruhi kemampuan untuk memenuhi atribut kualiti seperti kebolehubahan, prestasi dan sekuriti. Adalah penting untuk mampu menganalisa dan menaakul mengenai seni bina tersebut. Walau bagaimanapun dokumentasi seni bina kadang kala wujud, dan bila ianya wujud, ianya tidak selaras dengan sistem yang diimplemen. Tambahan pula tidak semua pembangunan perisian bermula dengan keperluan baru sepenuhnya; sistem dikekang oleh kewujudan kod legasi. Akibatnya kita perlu mampu menghasilkan maklumat dari implementasi sistem yang sedia ada dan menaakul secara seni bina mengenai maklumat ini.

Penyelidikan ini mempersembahkan satu alatan kejuruteraan songsang VOO++ yang dapat membaca kod sumber berorientasi objek C++ dan menghasilkan notasi

UML untuk mengvisualisasi struktur kelas dan pelbagai hubungan yang mungkin wujud termasuk pewarisan, agregasi dan hubungan kebergantungan berdasarkan kepada algoritma perubahan kepiton Cohen-Sutherland. Idea kepiton disongsangkan, sebagai gantian kepada kepiton dalam segi empat, kepiton dilakukan di luar segi empat dalam rangkaian empat arah (kiri, kanan, atas, bawah) dan dua titik mewakili titik tengah untuk setiap empat segi.

Pendekatan berorientasikan objek digunakan untuk mereka bentuk dan mengimplemen alatan tersebut. Teknik kejuruteraan songsang, corak reka bentuk dan grafik digunakan sebagai asas. VOO++ membantu penganalisa dalam menghasil, memanipulasi dan menginterpretasi maklumat statik model berorientasikan objek. Dengan membantu dalam membina semula seni bina statik dari maklumat yang terhasil, VOO++ menolong penganalisa untuk mendokumen semula dan memahami seni bina dan mengetahui hubungan diantara seni bina “yang diimplemen” dan “yang direka bentuk”.

ACKNOWLEDGEMENTS

In the name of Allah, Most Gracious, Most Merciful

I would like to take this opportunity to convey my sincere thanks and deepest gratitude to my chairman supervisor Dr. Abdul Azim Abdul Ghani for his advises, comments, suggestions, help, and invaluable guidance, and fruitful discussions throughout my research.

I am also indebted to Dr. Ramlan Mahmud and Dr. Md. Nasir sulaiman, members of the supervising committee, for their technical support, suggestions and insights are priceless.

I am greatly indebted to the Libyan Ministry of Education for financial support during my study. The contribution of the people at the Libyan Bureau is highly appreciated.

I owe the biggest debt to Engineering Academy and my family for their assistance and support. They are the biggest contributors to my success.

Finally, special mention must be made of all my friends at faculty of computer science and information technology without their knowledge this work would not have been done. These people deserve to be recognised and so I want to do, but I am afraid if I do, I will definitely miss some names.

I certify that an Examination Committee met on 5th October 2000 to conduct the final examination of Ali Hussein A. Mresa on his Master thesis entitled "Extracting Object-Oriented Software Architecture from C++ Source Code" in accordance with Universiti Pertanian Malaysia (Higher Degree) Act 1980 and Universiti Pertanian Malaysia (Higher Degree) Regulations 1981. The committee recommends that the candidate be awarded the relevant degree. Members of the Examination Committee are as follows:

Md. Yazid Mohd Saman, Ph.D.
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Chairman)

Abdul Azim Abd. Ghani, Ph.D.
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Member)

Ramlan Mahmod, Ph.D.
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Member)

Md. Nasir Sulaiman, Ph.D.
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Member)



MOHD. GHAZALI MOHAYIDIN, Ph.D.
Professor/Deputy Dean of Graduate School
Universiti Putra Malaysia

Date: 12 OCT 2000

This thesis was submitted to the Senate of Universiti Putra Malaysia and was accepted as fulfilment of the requirements for the degree of Master Science.



KAMIS AWANG Ph.D.

Associated Professor/Dean of Graduate School
Universiti Putra Malaysia

Date: 11 JAN 2001

DECLARATION

I hereby declare that the thesis is based on my original work except for quotations and citations, which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UPM or other institutions.



Name: Ali Hussein A. Mresa

Date: 10-10-2000

TABLE OF CONTENTS

	Page
DEDICATION	II
ABSTRACT	III
ABSTRAK	V
ACKNOWLEDGEMENTS	VII
APPROVAL SHEETS	IX
DECLARATION	XI
LIST OF TABLES	XVI
LIST OF FIGURES	XVII

CHAPTER

1	INTRODUCTION	1
	1.1 Background	1
	1.2 Software Architecture	2
	1.3 Importance of Software Architecture	2
	1.4 Software Architecture Issues	3
	1.5 Research Objectives	5
	1.6 Thesis Organisation	6
2	OBJECT-ORIENTED SOFTWARE DESIGN: PRINCIPLES, BENEFITS, ARCHITECTURE AND REVERSE ENGINEERING TOOLS	8
	2.1 Introduction	8
	2.2 Object-Oriented Design.....	8
	2.3 Principles of Object-Oriented Software.....	10
	2.3.1 Objects	10
	2.3.2 Classes	11
	2.3.3 Abstraction	11
	2.3.4 Encapsulation	12
	2.3.5 Polymorphism.....	13
	2.3.6 Modularity	13
	2.3.7 Hierarchy.....	14
	2.4 Benefits of Object-Oriented Approach.....	16
	2.5 Software Architecture.....	16
	2.6 Architectural Structures	20
	2.7 Object-Oriented Software Architecture	22



2.8	Object-Oriented Methods	23
2.8.1	Booch Method	23
2.8.2	OMT Method	24
2.8.3	OOSE/Objectory Method	25
2.8.4	OORA Method	25
2.8.5	Unified Modelling Language	25
2.9	Reverse Engineering	28
2.9.1	Redocumentation	30
2.9.2	Design Recovery	30
2.9.3	Reverse Engineering Purposes	31
2.10	Reverse Engineering Tools	32
2.10.1	Dali	32
2.10.2	CC-RIDER	36
2.10.3	Imagix 4D & Imagix 2000	40
2.10.4	Extracting and Preserving Low-Level Program	43
2.10.5	Rational Rose Case Tool	44
2.11	Characteristics of Reverse Engineering Tools	47
2.12	Summary	49
3	OBJECT-ORIENTED METHODOLOGY	50
3.1	Introduction	50
3.2	Object-Oriented Development Methodology	51
3.3	Object-Oriented Development Inputs	53
3.4	Methodology Steps	54
3.4.1	Use Cases	54
3.4.2	Develop Message Flow Diagrams	56
3.4.3	Develop Collaboration Diagrams	57
3.4.3.1	Identify Classes	59
3.4.3.2	Identify Class Attributes	61
3.4.3.3	Identify Responsibility	61
3.4.3.4	Identify Subsystems	61
3.4.3.5	Identify Contracts	62
3.4.4	Develop Hierarchy Diagrams	63
3.5	Summary	63
4	VOO++ SOFTWARE DESIGN AND IMPLEMENTATION...	66
4.1	Introduction	66
4.2	Business Phase	67
4.2.1	Prerequisites	67
4.2.2	Activities	67
4.2.3	Deliverables	68
4.2.3.1	VOO++ Functional Requirement	69
4.2.3.2	VOO++ Non-Functional Requirement	69

4.3	Analysis Phase	70
4.3.1	Prerequisites	71
4.3.2	Activities	71
4.3.2.1	Write Use Cases	72
4.3.2.2	Extract Noun List From Use Cases And Initial Requirements Documentation	75
4.3.2.3	Identify and Document Application Classes from the Noun List	77
4.4	Design Phase	79
4.4.1	Prerequisites	81
4.4.2	Activities	81
4.4.2.1	Identify Classes and its Responsibilities ...	81
4.4.2.2	Search the Reuse Library for Applicable Components	85
4.5	Implementation Phase	86
4.5.1	Document the Target Language, Hardware, and Software Platforms	86
4.5.2	Major Data Structures In VOO++ Application	87
4.5.2.1	Class Table	87
4.5.2.2	Class Table Relationships	91
4.5.2.3	Data Base	92
4.5.3	Major Graphics Techniques of VOO++ Application	94
4.5.4	Mathematics Preliminaries	96
4.5.5	Modified Cohen-Sutherland Clipping Implementation	98
4.5.6	Class Relationships Implementation	103
4.6	User Interface	107
4.7	Summary	111
5	RESULTS AND DISCUSSION	113
5.1	Introduction	113
5.2	Bill-of-Material Case Study	114
5.2.1	Bill-of-Materials Classes	116
5.2.2	Bill-of-Materials Aggregation Relationships	117
5.2.3	Bill-of-Materials Inheritance Relationships	118
5.2.4	Bill-of-Materials Dependency Relationships	119
5.2.5	Bill-of-Materials Class Diagrams	120
5.3	Buffer-Module Case Study	124
5.3.1	General Class Specification	125
5.3.2	Operations Class Specification	128
5.3.3	Attributes Class Specification	128
5.3.4	Member List Class Specification	130
5.3.5	Reports	131

5.3.5.1	Logical View Report	131
5.3.5.2	File Summary Report	133
5.3.5.3	Class Summary Report	133
5.4	Documentation Consequence	133
5.5	Summary	135
6	CONCLUSION AND FUTURE WORK	137
6.1	Conclusion	137
6.2	Future Work	140
	REFERENCES	141
	APPENDIX	147
A	UML Notation	148
B	Complete VOO++ Software Application	153
C	Summary of the C++ language (Subset of C++)	181
D	Buffer-Module Case Study Source File	185
	VITA	193

LIST OF TABLES

Table		Page
5.0	TListBuffer Class Operations Specification	128
5.1	TListBuffer Class Attributes Specification	130
5.2	TListBuffer Class Members List	130
5.3	VOO++ File Summary Report of Bill-of-Materials.....	133
5.4	VOO++ Class Summary Report of Bill-of-Materials.....	133

LIST OF FIGURES

Figure	Page
2.0 Communication Between Objects	9
2.1 Dali Workbench	34
2.2 A Derived Relationships	35
2.3 CC-RIDER Architecture	37
2.4 File Used Tree CC-RIDDER Sample Output	39
2.5 Class Hierarchy CC-RIDDER Sample Output	39
2.6 File Structure (Imagix)	42
2.7 Class Structure (Imagix)	43
2.8 Module Diagrams Using UML Notation	46
2.9 Class Diagrams Using UML Notation	47
3.0 Object-Oriented Development Methodology	51
3.1 Inheritance Use Case	55
3.2 Scan Use Case	57
3.3 Part of Visualisation Subsystem of VOO++ Application	58
3.4 VOO++ Subsystems.....	62
4.0 Analysis Phase Prerequisites and Deliverables	72
4.1 Basic Architecture for Reverse Engineering, Reengineering Tools.....	76
4.2 Document/View Interface	76
4.3 VOO++ Primitive Classes	77
4.4 VOO++ Collaboration Diagrams	79
4.5 Design Phase Prerequisites and Deliverables	80
4.6 Class Table Implementation	89
4.7 Class Table Dynamic Structure	90
4.8 Class Table Relationships Implementation	91
4.9 Class Table Relationships Structure	91
4.10 DataBase Structure	93
4.11 DataBase Implementation	93
4.12 M and N relationship	95
4.13 Start and End Points Relationship	96
4.14 The Slope Relation	97
4.15 Two Equality Cases	100
4.16 Relationships Structure and Notation	103
4.17 Shapes Generation	106
4.18 VOO++ Main Window	107
4.19 VOO++ PopUp and Bar Menus	108

4.20	VOO++ About Dialog	109
4.21	VOO++ Context Menu	110
4.22	VOO++ Report Dialog	111
5.0	Bill-of-Material Classes Definitions	115
5.1	Bill-of-Material Classes Rectangles	116
5.2	Bill-of-Material Aggregation Relationships	117
5.3	Bill-of-Material Inheritance Relationships	119
5.4	Bill-of-Material Dependency Relationships	120
5.5	Bill-of-Material Static Model	121
5.6	Buffer-Module Classes Definitions	122
5.7	Buffer-Module Inheritance Relationships	124
5.8a	Buffer-Module General Class Specification	126
5.8b	Buffer-Module General Class Specification	126
5.8c	Buffer-Module General Class Specification	127
5.8d	Buffer-Module General Class Specification	127
5.9	TListBuffer Class Operations Specification	129
5.10	TListBuffer Class Attributes Specification	129
5.11	TListBuffer Class Members List	131
5.12	VOO++ Logical View Partial Report	132
5.13	Bill-of-Material as Designed	134
A.1	Class Rectangle	148
A.2	Object Rectangle	148
A.3	Dependency Relationship	149
A.4	Association Relationship	149
A.5	Unidirectional Association Relationship	149
A.6	Aggregation Relationship by Reference	150
A.7	Aggregation Relationship by Value	150
A.8	One to One Multiplicity	150
A.9	One to Many Multiplicity	150
A.10	Inheritance Relationship	151
A.11	Multiple Inheritance Relationship	152
A.12	Sequence Diagrams	152
B.1	Open Document Sequence Diagram	153
B.2	Scan Document Sequence Diagram	154
B.3	Inheritance Sequence Diagram	155
B.4	Dependency Sequence Diagram	156
B.5	Aggregation Sequence Diagram	156
B.6	Application Classes Sequence Diagram	157
B.7	Class Diagrams Sequence Diagram	158
B.8	Class Functionality Sequence Diagram (First Scenario)	158
B.9	Class Functionality Sequence Diagram (Second Scenario)	159
B.10	Class Attributes Sequence Diagram (First Scenario)	160
B.11	Class Attributes Sequence Diagram (Scenario)	160
B.12	Class Specification Sequence Diagram (First Scenario)	161

B.13	Class Specification Sequence Diagram (Second Scenario)	162
B.14	Class Functions and Attributes Sequence Diagram	163
B.15	Mouse Facilities Sequence Diagram	164
B.16	General Report Sequence Diagram	165
B.17	Logical View Report Sequence Diagram	166
B.18	Statistics Model Report Sequence Diagram	167
B.19	File Summary Report Sequence Diagram	168
B.20	Class Summary Report Sequence Diagram	169
B.21	VOO++ Collaboration Diagram—(Module Architectures)	170
B.22	VOO++ Collaboration Diagram—Extraction Subsystem	171
B.23	VOO++ Collaboration Diagram— Data Base Subsystem	172
B.24	VOO++ Collaboration Diagram— Visualisation Subsystem	173
B.25	VOO++ Application Reused Classes	176

CHAPTER 1

INTRODUCTION

“If a project has not achieved a system architecture, including its rationale, the project should not proceed to full-scale system development. Specifying the architecture as a deliverable enables its use throughout the development and maintenance process” (Boehm, 1995).

1.1 Background

Architectural design has always played a strong role in determining the success of complex software-based systems, the choice of an appropriate architecture can lead to a product that satisfies its requirements and is easily modified as new requirements present themselves, while an inappropriate architecture can be disastrous (Garlan, 1997; Buxton & McDermid, 1991).

Despite its importance to software systems engineers, the practice of architectural design has been largely ad hoc, and informal. As a result, architectural designs are often poorly understood by developers; architectural choices are based more on default than solid engineering principles; architectural designs cannot be analysed for consistency or completeness; architectural constraints assumed in the

initial design are not enforced as a system evolves; and there are virtually no tools to help the architectural designers with their tasks (Garlan, 1997).

1.2 Software Architecture

Software architecture concerns the structures of large software systems. The architectural view of a system is an abstract view that distils away details of implementation, algorithm, and data representation and concentrates on the behaviour and interaction of "black-box" components. Software architecture is developed as the first step toward designing a system that has a collection of desired properties. Also the product of software design activities are the definition of the software architecture specification.

1.3 Importance of Software Architecture

Fundamentally, there are three reasons why software architecture is important, as follows (Bass *et al.*, 1998).

- Communication among stakeholders. Software architecture represents common high-level abstraction of a system that most if not all of the system's stakeholders can use as a basis for creating mutual understanding, forming consensus, and communicating with each other.

- **Early design decisions.** Software architecture represents the manifestation of the earliest design decisions about a system, and these early bindings carry weight far out of proportion to their individual gravity with respect to the system's remaining development, its deployment, and its maintenance life. It is also the earliest point at which the system to be built can be analysed.
- **Transferable abstraction of a system.** Software architecture constitutes a relatively small, intellectually graspable model for how a system is structured and how its components work together this model is transferable across systems; in particular, it can be applied to other systems exhibiting similar requirements and can promote large-scale reuse.

1.4 Software Architecture Issues

The formal study of software architecture has been a significant addition to the software-engineering repertoire in the 1990s. It has promised much to designers and developers: help with the high-level design of complex systems. Early analysis of high-level designs; particularly with respect to their satisfaction of quality attributes such as modifiability, security, and performance; higher level reuse such as that of designs and enhanced stakeholders communication (Garlan, 1993). These benefits seem enticing. However, much of the promise of software architecture has as yet gone unfulfilled. Some of the problems simply stem from the fact that architectures are seldom documented properly (Kazman & Carriere, 1997) where:

- Many systems have no documented architecture at all. (All systems have an architecture, but frequently it is not explicitly known or recorded by the developers and therefore evolves in an ad hoc fashion.)
- Architectures are represented in such a way that the relationship between the architectural representation and the actual system, particularly its source code, is unclear.
- In systems that do have properly documented architectures, the architectural representations are frequently out of sync with the actual system, because maintenance of the system occurs without a similar effort to maintain the architectural representation.
- There is little completely new development. Development is typically constrained by compatibility with, or use of, legacy systems. And it is rare that such systems have an accurately documented architecture. Because of these issues, a serious problem in assessing architectural conformance arising, which makes system understanding and maintaining unbearable. Mismatch between “as designed” and “as implemented” system architectures cause much of the value of having an architecture is lost.

In addition, when a system enters the maintenance phase of its life cycle, it may sustain modifications that alter its architecture. Hence, a second problem arises: how to ensure that maintenance operations are not eroding the architecture, breaking down abstractions, bridging layers, compromising information hiding, and so forth?

All of these are manifestations of two underlying causes. The first is that a system does not have “an architecture”. It has many: its runtime relationships, data flows, control flows, code structure, and so on. The second, more serious, cause is that the architecture that is represented in a system’s documentation may not coincide with any of these views.

1.5 Research Objectives

With reference to the software architecture issues the research is aiming to develop a prototype reverse-engineering tool base on an Object-Oriented approach with reverse engineering underlying. The proposed tool called VOO++ (Visualisation of Object Oriented Architecture from C++ source code) that helps the stakeholders to:

- Re-document Object-Oriented software architecture using an Object-Oriented reverse engineering tool, which implies using standard notations UML (unified modelling language) to facilitate the communications between project teams. As a result of documentation, the relationship between “as designed” and “as implemented” system architecture is presented, which is provided that the software under analysis should be documented during its development process.

However, the research undertaken will cover the following: